PORTING OF PVM ON WINDOWS 95

By
B.M. Shukla

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

AUGUST, 1997

Porting of PVM on Windows 95

A Thesis Submitted

in Partial Fulfilment of the Requirements

for the Degree of

Master of Technology

By
B.M. Shukla

to the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

AUGUST,1997

CERITIFICATE

This is to certify that the work contained in the thesis titled **Porting of PVM on Windos95** by B.M. Shukla has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr. R. K. Ghosh
Dept. CSE,
IIT Kanpur

Acknowledgments

I am very thankful to my supervisor Dr. R.K. Ghosh. I appreciate Dr. Ghosh for his firmness, helping hand , target orientation and commitment for the work. I am very thankful to Dr. Tapas Nayak for providing fruitful guidelines to this work.

I am thankful to All India Counsel of Technical Education, New Delhi (AICTE/CSE/94205) for providing financial assistance.

Special thank to my friends NV Brahmaji Rao, Aniruddha Mukhopadhyaya and Devashish Roy Chowdhury, for providing me morel support.

I am thankful to my friend Amitabh Roy, for providing me required technical help.

Thanks to the faculty of CSE department IIT Kanpur for providing me adequate help.

A lovely thank to my wife Sudha and my kid Sulochana for their kind cooperation during this period of time.

This thesis is dedicated to my parents, Ravana and Shiv, my sources of inspiration.

Abstract

PVM is a network orchestration tool for low cost useful parallel computing. It was only available to most of the UNIX systems. After this successful port on Windows 95, small and general users will be benefited. Now a person (or an organization) can enjoy parallel computing using Windows95.

Porting of PVM over Windows95 environment is a typical example of porting a UNIX software to a PC platform. This port provided framework for porting other UNIX network software to Windows95 Platform. This framework provide now OS independent software can be written. It can become a teaching tool for parallel computing. In India now schools and colleges also can have a practical look and feel of parallel computing.

WinPVM can make use of all those under utilized PCs, which are only use for text processing type jobs. PCs are used as single user system and most of the time these machine are idle (CPU idle time). Using WinPVM this idle time can be utilized effectively. PCs are not as costly as UNIX servers. Also upgrading to faster hardware is easy and cheap in case of PCs.

It supports heterogeneity in many dimension including architecture, data format, computational speed, machine load and network load etc. However PVM requires OS level support of either UNIX or UNIX clones. The main aim of this work is to investigate the issues which need to resolved so that the network orchestration tools like PVM or MPI can also support heterogeneity in OS. The other motivation is to effective utilization of idle CPU time of personal computers accessible through network.

Contents

Chapter 1

Introduction

Large scale computational problems in weather forecasting, computational fluid dynamics, protein synthesis, etc., require tremendous CPU speed which can only be met using parallel/distributed computing. The closely coupled parallel processing systems like MPP can have between hundred to few thousand CPUs with gigabytes of RAM. However, using such huge monolithic systems is cost prohibitive. Only a few government organizations or a only few Fortune 500 based companies may afford such costly hardware solutions to compute-intensive problems. Besides the cost, the closely coupled systems have another drawback, viz., upgradebility. Suppose with improved hardware technology cheaper and more powerful chips become available. One may like to upgrade the system taking the advantage of cheaper hardware. This may not be a simple job in case of MPP like systems. Perhaps the entire system may have to be reengineered to make the upgradebility meaningful in terms of performance.

There are also some software linked issues which need to be addressed for tuning performance of closely coupled parallel processing systems. A high level language with easy to use interface for exploiting parallel processing features should be available to reduce the challenges on programming skill of end users and increase their productivity. Optimized compilers for such languages must also have to be installed to exploit inherent

concurrency of machine architecture. From commercial angle the interests in above mentioned software issues have only been confined to academic and research institutes.

Practical approach to cater to computational requirements of large scale problem is to seek for a software based open architecture solution. Several such options like P4 [2], Express [3], MPI [4], Linda [5] and PVM [1] are available. A quick review of these systems is provided in following sections.

1.1 P-4

P4 [2] system is meant for distributed as well as shared memory models. It uses message passing system for communication among nodes for distributed memory model. P4 provides operations such as send, receive and process creation by probing a configuration file. The configuration file also describes process groups and process structures. For shared memory model, P4 provides a set of monitors and a set of primitives.

Using information available in configuration file, a machine pool is created. Every machine of the pool executes an executable P4 program. The master-slave paradigm is used for processing. The master will collect results after processing on each slave node is over.

It also supports multilevel hierarchies to implement cluster model of computing. The primary mode of process creation is static but dynamic process creation is also supported through configuration file. Both blocking and asynchronous transfers are supported. P4 also supports broadcasting and barrier synchronization.

1.2 Express

Express [3] is another software solution for parallel processing. Commercially, express is marketed by a company called Parasoft Corporation. It is basically a toolkit which supports various aspects of concurrent processing. It consists of tools such as VTOOL, FTOOL and ASPAR.

VTOOL is a graphical interface which shows progress of sequential algorithm in a dynamic manner. It provides detailed information about hidden parallelism in sequential algorithm. FTOOL performs a program analysis to provide the flow structure and feedback about would be parallelization. The third tool in this series is ASPAR. It is an automated parallelizer that restructures a sequential program in such a way that it can exploit facilities for parallel or distributed execution. This parallelizer accepts programming languages C and FORTRAN.

Express uses set of libraries for I/O communication and parallel graphics. It includes a variety of global operations and data distribution. The communication primitives are akin to those found in other message passing system. Extended I/O routines allow parallel input and output. Similar set of routines are also available for graphical display from multiple concurrent processes. Express uses a parallel debugger tool named NDB. It uses compatible set of commands as found in UNIX dbx.

1.3 MPI (Message Passing Interface)

MPI [4] is a specification which deals with syntax and semantics of a message passing library routines. It enforces a standardization in message passing interface. So

MPI programs are portable across platform. There are many MPI implementations over wide range of MPPs and other distributed network machines. A standard implementation uses RPC (Network Interface Layer Developed By Sun) in place of socket interface for network communication. It does not includes process management. So MPI sits as a communication interface layer which effectively uses underlying hardware. Of course some of the data transfer operations can be implemented close to hardware level. But such hardware dependencies require MPI to be implemented afresh for different set of hardware.

1.4 Linda

Linda [5] is a concurrent programming model evolved by Yale University research Project. From application point of view Linda can be viewed as a programming language extension (like PVM) which supports parallel programming. It provides shared memory abstraction without a physical shared memory hardware for process communication.

It provides a library which contains primitives for Linda system. It works in different environment like shared memory multiprocessors system, message passing parallel computers, a platform having various workstations connected with network. In all environments stated above implementations vary in efficiency.

1.5 PVM (Parallel Virtual Machine)

PVM is developed basically for heterogeneous UNIX based platforms. PVM stands for parallel virtual machine. PVM creates a virtual machine out of heterogeneous

UNIX based platforms connected by a network. For a huge computational task the time of execution can be reduced drastically using PVM and distributed computing. For small jobs the network over heads may result a poor response from PVM.

According to our information there have been very little effort on supporting heterogeneity of O/S in PVM implementations. In [13] a design of PVM for Windows 3.1 has been proposed. It was also claimed that an implementation of WPVM has been done. However, our repeated effort to get the details from the authors did not succeed. Besides, this work came to our notice after we have reached half way stage with the present implementation.

Only a couple months back while browsing http site a Windows NT port for PVM also came to our notice. Therefore, only other contemporary effort for supporting heterogeneity of O/S for PVM implementation known to us is by the principal authors of PVM. At that moment 95% port of PVM was over. However, we did not try to install NT version as we believe it may not work, as fork() system call which is not supported by WIN32 must have to be replaced by semantically equivalent function.

Using our version Win95PVM one can make a desktop machine running Windows95 as master and other nodes running UNIX or Windows 95 as slave PVM nodes. Similarly from a UNIX workstation the master pvmd node can attach both Windows 95 and other UNIX based systems simultaneously.

So it provides complete heterogeneity between both Windows 95 and UNIX based systems for constructing a parallel virtual machine.

1.6 Motivation

PVM is a low cost network based distributed computing tool. It uses available UNIX servers and workstations connected over a LAN or WAN. It uses standard TCP/IP based socket communication exchange of data and information. Only a few minimum set of PVM primitives are provided for writing programs. The programmers use the embedded PVM calls in programs. A make file with minimum changes can be used to compile PVM program with help of standard PVM library.

PVM for UNIX platforms succeeded in reducing parallel computing cost by using available systems connected by LAN. Yet for economic reasons in developing country like India no organization can afford to buy number of UNIX platforms. Typically in academic institutions and universities, one finds just one or two UNIX servers and perhaps a few workstations. However, most organizations can afford a few hundred PCs and PC based servers.

Apart from cost considerations, UNIX Server management is not an easy job. Though Linux is available on PCs, due to management problems, a general user would prefer a more friendly environment like Windows95.

The other motivation behind this project was to use idle time of CPUs for parallel computing, specially PCs. Normally PCs are used as desktops for text processing kind of applications. Such applications are not CPU intensive jobs. Now a days Pentium -200 MHz PCs are available at very low cost. Therefore, such PCs on LAN may contribute tremendous amount of idle CPU time to run PVM based application programs in background. All these factors motivated this present work for porting PVM on Windows95 platforms. We expect that this porting could prove to be an useful teaching

tool for parallel computing. The only required investment is for networking as hardware, if not available.

1.7 Thesis Overview

This thesis is organized into five chapters including this introductory chapter.

Chapter 2 gives a brief description about design of PVM for UNIX platforms. It also focuses on the main features of Windows 95. Chapter 3 deals with standard for running software developed on UNIX to Windows95 environment porting issues. It describes about various types of requirements for code restructuring. It tells about differences as well as non availability of UNIX like functionalities and libraries in Windows95 environment. The major problem in porting was due to the fact that the functional requirements for implementing UNIX system call fork() and pipe() could not be exactly met by Windows95 environment. Chapter 4 deals with complete design and implementation of WinPVM highlighting basic departure from original with PVM design and implementation. The conclusions have been drawn up in chapter 5.

. Apart from 5 chapters this thesis also include 2 appendices. Appendix1tells about setup & configurations for WinPVM. Appendix 2 tells about restructuring of existing PVM based application programs for WinPVM.

Chapter 2

PVM (Parallel Virtual Machine)

PVM is a Unix based software used for parallel computing. It uses several heterogeneous Unix workstations, connected through network, as a single virtual machine. It uses TCP/IP as communication layer.

PVM supports Heterogeneity in

- Architecture
- Data format
- Computational speed
- Machine Load
- Network load

This chapter provides a quick review of PVM design and implementation issues in prospective of those encountered while porting it to Windows95 environment.

2.1 Heterogeneity in architecture

PVM supports number of Architectures. Some typical architectures are DEC-Alpha, AlphaMP, HP300, HPPA, SUN4, SUNMP. For complete list of architectures supported by PVM the reader may refer to [1].

Though PVM implementation provides a uniform interface for different UNIX versions on wide range of machine architectures, it does not support OS heterogeneity.

For instance, PVM runs on Intel 486/Pentium architectures for BSD386 or Linux. However, DOS or Windows or Windows 95 are not supported for same Intel machine architectures.

2.2 Heterogeneity in Data Formats

PVM supports XDR library conversion for different data formats. The data sent from any machine to another machine supporting a different data format gets converted to network data form (Network Byte Order). At destination network data format is converted into the target machine's data format. Such conversion is needed if source and target machines have different data format.

If source and destination machines have same data format, then XDR conversion is not done in PVM. So data format conversion overhead is avoided for homogenous machine architectures.

2.3 Heterogeneity in Computational Speeds

A number of physical machines connected over network constitute the parallel virtual machine. There may be wide variation in computational power of these machines. For instance, a Pentium@200 MHz is at least 5 times faster than a 486@66 MHz. PVM takes this heterogeneity in computational speeds of physical machines in to account while distributing tasks.

Each member of virtual parallel machine can be assigned a weight for its computational capabilities relative to the member which has the least weight.

2.4 Heterogeneity in Machine Load

In parallel virtual machine, the constituent nodes may have different architectures, different computational speed and different system loads at different instant of time. The system load of the node can be quantized in a similar manner as its computational speed. A powerful node need not be lightly loaded. It is likely that there is a greater demand for this node compared to other nodes in the virtual machine as users expect a fast response from such a node. PVM probes system loads of various nodes before assigning tasks.

2.5 Advantages of using PVM

A clear advantage of using PVM is in its abilities to exploit multifold heterogeneity of an existing network of computers.

It has several other additional advantages. It runs over existing hardware. That is, it does not require any installation of new network. It allows for an open architecture. Individual system upgradation immediately enhances performance. Similarly, it can absorb any new development in network technology for linking nodes.

The power of PVM depends on CPU power of the machine as well as the implementation of network layer. If the technology development lead to an improvement of any of these two a better performance results. Adding new machines to network does not require any change in PVM software. Suppose one installation has Ethernet supports peek data transfer rate of 10 Mbits/Sec. If this installation decides to switch to a high speed network FDDI (Fiber Distributed Data Interface having speed 100-Mbits/sec) or HiPPI (High Line Performance Parallel Interface having 800 Mbits/Sec over 32 parallel

line or 1.6 Gbits/Sec over 64 parallel lines), the performance of PVM improves tremendously.

PVM provides environment for computing only. It does not use its own compiler, debugger and editors. One can use editors, compilers and debuggers of his own choice.

2.6 PVM System Overview

PVM is a software consists of a console program, a PVM daemon program and a library. The library called Libpvm supports primitives for construction of a virtual machine having different architecture, different network (interconnected systems only). PVM provides complete freedom to construct a virtual machine. Nodes may be added to virtual machine, or removed from it at any time

Unit of parallelism in PVM is a process. The processes can execute in parallel . This unit is not always similar to a UNIX like process. Sometimes it is a thread of execution and sometime it is a complete process. PVM refers to this unit as task

PVM can also take the advantage of multiprocessors system . However tailored and optimized copy of PVM software is required for such system. A few known multiprocessor versions of PVM already exist. They can communicate in a uniform manner with other ordinary PVM.

2.7. PVM modules

PVM has three major modules and some optional modules also. The main modules are

- PVM Console (pvm)
- PVM Daemon (pvmd /pvmd3 both are alias to each other)
- PVM library (libpvm)

2.7.1 PVM Console:

PVM console is a program which helps in creating and configuring the virtual machine. Some useful console commands are described below.

add Adds new host to the virtual machine

alias Sets alias for command set and displays of alias list

conf Displays configuration of virtual machine

delete Removes a node from current virtual machine.

halt Shuts down all pvm processes, then daemons (pvmd) and finally kills the console program.

help Displays complete command set for PVM. "*help <command>*" will display the help for *<command>*

quit Quits PVM console only. PVM daemon still runs.

version Tells about PVM version running on the console node (computer).

There are several other commands also (See [1]).The console commands are quite simple to use.

From one application to another, the configuration of virtual machine may change. There is a better way to get around the problem of configuration. A separate host configuration file may be used for this purpose. PVM Daemon reads the host configuration file and starts the preconfigured slaves. All aliased commands and

environment setting commands can similarly be put in a file called pvmrc in home directory.

2.7.2 PVM Daemon

The next major module is PVM Daemon. It is the heart of PVM software. PVM Daemon act as a message router and controller. It provides authentication, process control, fault detection/notification. Time to time is also checks about the health of the virtual machine probing the constituent nodes whether they are alive or not (Keep alive message). PVM uses a task id (*tid*) to identify the pvmids (Master or slave), the tasks and the group of tasks within a virtual machine. One PVM daemon runs on each node of a virtual machine. But in one physical machine several PVM daemons can run concurrently. Each of these PVM daemon belongs to only one virtual machine setup. That is though several PVM daemons can coexist in a single machine (for various different virtual machines owned by different users), but they do not interact with each other. In one virtual machine one node act as Master PVM Daemon (from where PVM console program starts) and others behave as slave. There is hardly any difference between slave and master. They are identical in sense of PVM though not in speed. Basically nodes emulate to provide the Mandelbrot computation paradigm. However, PVM console can be run from only master PVM Daemon. Only Master PVM daemon has capacity to manage other nodes with the help of PVM console program.

2.7.3 Network Protocols used by PVM

PVM communication is based on TCP/IP based socket. It uses TCP as well as UDP for communication. In Windows95 only Winsock library is used.

Between one PVM daemon to Other PVM daemon PVM uses UDP sockets. UDP socket do not provide connection oriented service. However, due to extra overhead in using TCP this decision has been taken. If packet lost in between, then retransmission and acknowledgment policy is used.

Between Pvmd Daemon and PVM task TCP socket is used due to reliability of TCP protocol. Similarly to achieve reliability in network communication between PVM task to other PVM task, TCP sockets are used. Only one major drawback is in using TCP socket is overhead. As we know for reliability one has to pay cost in overhead sense. Example of over head is that one can send a packet using one **send** socket call but the same packet will be received at other end by calling **recv** socket call twice. It first receives header of the message by one call. In this header it will indicates how much more data is left for reading. After getting this information again **recv** system call is executed to get complete data.

2.7.4 PVM Library (LibPvm):

PVM library is a very important part of PVM. It provides to the interface of PVM programming. For details of each primitive and its functionality the reader may refer to [1].

2.8 Windows 95

Windows 95 is a 32 Bit operating system. Windows 95 unlike Windows 3.1 does not run over DOS. DOS runs as a shell on the top of Windows95. Several DOS shells may run simultaneously on a Windows95 environment.

Windows95 is a multi threaded, preemptive multitasking system (Similar to UNIX), i.e. ,one can run several threads simultaneously , which was not possible with Windows 3.1.

Most of the OS code is written in using 32 bit instructions, so it can take the advantage of Intel 386/486/pentium machine architecture. Windows 95 has 32 bit memory manager, scheduler and process manager. Some of the code is in 16 bit for downward compatibility with Windows3.1 and DOS.

It supports long filenames (255 characters UNIX like).

For configuration of a Windows 95 system "**Registry**" is used. This registry can be accessed and modified from other remote machine (connected through network).

One PC (Windows95 as OS) can be used by several people. Each of them can keep his/her desktop, shared resources, user rights and other setting also. It is a type of authentication layer (As in UNIX) but this layer is not very strict. Windows95 does not have terminal server. Therefore, only one person can use this system.

It supports 32 bit installable file system, which in turn gives faster disk access.

It includes peer to peer networking with 32 bit network drivers. It also supports TCP/IP protocol along with Winsock. It supports latest plug and play. It avoids configuration problems.

One important level concept is missing in Windows 95, which is present in UNIX machines. That concept is about *fork()*. Windows95 supports thread but not *fork()*. In implementation level working of *fork()* is not same as of *CreateThread*()

Another difference with UNIX is in pipe implementation. In UNIX *pipe* is used along with select system call (select is used for asynchronous read/write notification).

In Windows 95 pipe implementation is there but there is no straightforward way for notification. So pipe can be used, but read/write through pipe will be a blocking call.

Chapter 3

Porting Issues

The task of porting software from one UNIX machine to an other UNIX machine though simple, may not be trivial at times. The data alignment, data format and several other parameters may have to be examined before porting. But by and large not major changes are required.

However porting software running on UNIX to Windows95 environment is not as straight forward. Some of the programming concept, operating system support and the standard library support vary significantly from UNIX to Windows95.

This chapter presents a brief but comprehensive study on all the porting issues. It gives a fair idea about the efforts that went into porting UNIX PVM to Windows95 PVM.

3.1 File/Data Format Conversion

A File name in DOS may consists of a maximum of eleven, of which eight first characters are for file name followed by a dot and three characters for extension name. Any porting UNIX software to DOS will first ensure that relevant files are renamed suitably to adhere to the above restrictions.

A line of text file in UNIX ends with <CR> only. However in case of DOS, it ends with <CR><LF>. This requires all text files be modified to insert an <LF> after every <CR>.

The standard header files (.h files) used in C programs running an UNIX are not exactly compatible to those available in MSVC++ Windows95. For example, in a typical network programming application on UNIX environment must include the following header files.

<fcntl.h>
<socket.h>
<netdb.h>
<netinet/inet.h>
<netinet/tcp.h>
<arpa/inet.h>
<sys/un.h>

These header files mentioned above are not available on Windos95 platform. Windows95 provides <winsock.h>. Therefore, network programming application that can switch transparently between UNIX and Windows95 environment must include the following macros.

```
#ifdef WINDOWS95
#include <winsock.h>
#else
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#endif
```

There are many such similar incompatibilities.

3.2 XDR Library and Missing functions

PVM is heavily dependent on standard library functions of UNIX environment. But many of these library routines are not available in Windos95. One such example is the *XDR library*.

The complete source code for XDR Library is available in ftp site of Sun Microsystems, Inc. This code was tailored according the PC architecture. For example length of integer, float, double etc. variables modified as per PCs specifications and compiled using MSVC++. This library can now be used for any other network applications including PVM .

Several functions such as gettimeofday(), ffs() etc. are not available in Windows95. All such functions were implemented along with other extra required functions organized in a separate file called extralib.c.

3.3 rsh and rshd

PVM uses many standard UNIX utilities like rsh, rexec, rshd etc.. None of these utility programs are available in Windows95 environment. These are some similar (but not compatible) shareware programs for Windows95 but due to problem of compatibility these utilities could not be used directly.

NCSA rsh code was written using one nonstandard *TCP/IP* library. So, this code was first converted into Winsock based code. It required lot of changes in original source code. Finally the modified rsh code is recompiled.

Example code for some changes done in original NCSA source code is shown below. As we see in original NCSA code a non standard TCP/IP based library calls are

used. After studying the logic of NCSA rsh source code, almost all the nonstandard calls were replaced by appropriate Winsock API calls. One such example is shown below.

```
#ifndef WINSOCK
  /* send the request */
  netwrite(conn_id, buff, ubase);
  if (debug) printf("netwrite succeeded--flushing buffer\n");
  netpush(conn_id);
#else
  if ((n = send(sockfd, buff, ubase, 0))> 0)
        if (debug) printf("Send succeeded %d bytes %s \n",n,buff);
#endif
```

For rshd standard *BSD 4.3* (Open Domain Source Code available for UNIX Machines) source. The source code is modified appropriately to run on Windows95. Though this code is not fully compatible with *UNIX* rshd, it suited our requirement. We gave name W95Rshd for this rshd program. One typical example of code replacement is use of *fork()*.

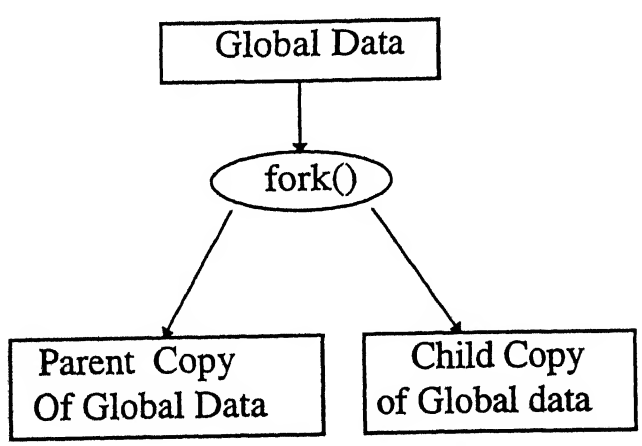


Fig. 3.1 fork()

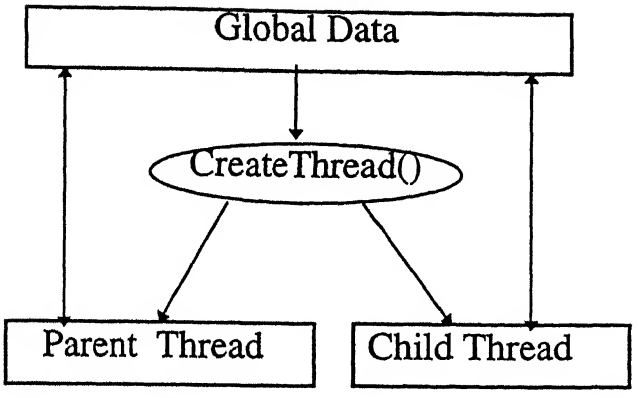


Fig. 3.2 CreateThread()

3.4 Internal Porting Issues

This section focuses on the nonportability of UNIX to Windows95 platform. The main issues are related to

- polymorphism in implementation of UNIX interface
- Absence of semantically equivalent Windows95 function for UNIX fork() call
- distinctions between functional requirements for interprocess communications of Windows95 and UNIX.

3.4.1 Polymorphism

Polymorphism or function overloading is frequently used by UNIX for manipulation of File, Pipe and Socket.

Four standard UNIX system calls *read*, *write*, *close*, *select* are used to manipulate *file*, *pipe* and *socket*. In windows95, *read*, *write* and *close* calls are only for file operations not for pipes and sockets.

The other major problem is in semantics of *select* system call. In UNIX select call is used uniformly for read/write notification in case of *file*, *pipe* and *socket*. In Windows95 *select* is available only for *socket notification* and not for *file* and *pipes*. Due to non availability of select for pipe we have to bear with blocking read and write functions for pipe operations. For example, read system call is to be used for network (socket) read, read call must be replaced with recv() winsock API function as given below.

```
#ifndef WINDOWS95
        n = read(tp->t_sock, pp->pk_dat + pp->pk_len, n);
#else
        n = recv(tp->t_sock, pp->pk_dat + pp->pk_len, n,0);
#endif
```

Similarly the equivalent winsock API function for network (socket) write is send(). Therefore write system call must be replaced as follows.

```
#ifndef WINDOWS95
        n = write(tp->t_sock, pp->pk_dat + pp->pk_len, n);
#else
        n = send(tp->t_sock, pp->pk_dat + pp->pk_len, n,0);
#endif
```

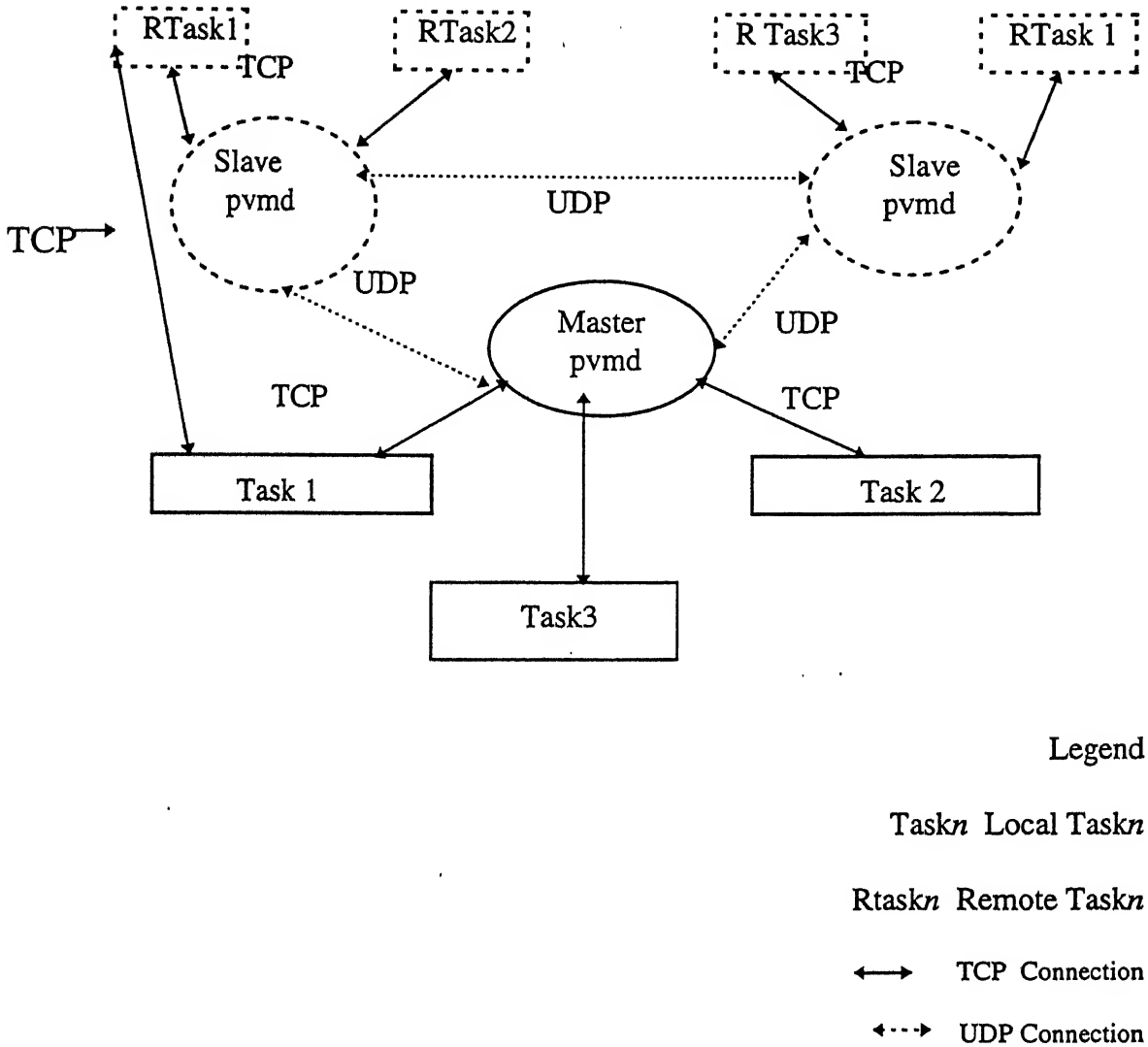


Fig 3.3:. Network Communication among

The close system call for close a socket should be replaced with closesocket() winsock API function.

Implementation of pipe is quite different in Windows95. If read system call is used for pipe read, replace this read system call with WIN32-APIs ReadFile() function. As shown in the example below.

```
#ifndef WINDOWS95
        n = read(sp->s_rfd, sp->s_buf + sp->s_len, sizeof(sp->s_buf) - sp->s_len);
#else
        ReadFile(sp->s_rfd, sp->s_buf+sp->s_len, sizeof(sp->s_buf)- sp->s_len,&n,NULL);
#endif
```

One point to note here is that n is going as parameter to ***ReadFile()*** while in case of *read()* (pipe read) it contains the return value of read system call. *ReadFile()* is a blocking call.

If write system call is used for pipe write, replace this write system call with WIN32-APIs *WriteFile()* function.

```
#ifndef WINDOWS95
        n = write(sp->s_rfd, sp->s_buf + sp->s_len, sizeof(sp->s_buf) - sp->s_len);
#else
        WriteFile(sp->s_rfd, sp->s_buf+sp->s_len, sizeof(sp->s_buf)- sp->s_len,&n,NULL);
#endif
```

If close system call is used to close pipe , replace this close system call with WIN32-APIs CloseFile() function.

```
#ifndef WINDOWS95
        n = close(sp->s_rfd);
#else
        CloseFile(sp->s_rfd);
#endif
```

There is one more major difference in pipe semantics. In UNIX *pipe* system call is used to create pipe. In Windows95 there is separate CreatePipe() function which creates a pipe. This function belongs to WIN32-API. So a pipe call should be replaced by CreatePipe() function as shown in the example below.

```
#ifndef WINDOWS95
        if (pipe(wpfd) == -1) {
                pvmlogperror("phase1() pipe");
                goto oops;
        }
#else
        // Set the bInheritHandle flag so pipe handles are inherited.
        saAttr.nLength = sizeof(SECURITY_ATTRIBUTES);
        saAttr.bInheritHandle = TRUE;
        saAttr.lpSecurityDescriptor = NULL;
        hSaveStdout = GetStdHandle(STD_OUTPUT_HANDLE);
        // Create a pipe for the STDOUT. [1]
        if (!CreatePipe(&hStdoutRd, &hStdoutWr, &saAttr, 0))
                ErrorExit("Stdout pipe creation failed\n");
#endif
```

Though most of these changes can be visualized as one to one mapping, but format for calling and call arguments may vary from function to function and system call to system call.

3.5 Simulation of Fork() call

In UNIX *fork()* is used to create new process. A close replacement for *fork()* system call in Windows95 could be *CreateThread()*. But, it can not be used as a true replacement of *fork()*. The parent and child created by *fork()* system call use separate (different) copies of global data as they have different address space. The child can not modify parent's global data and vice versa. The semantics of thread is different from a process. Though the threads have separate user stack the address spaces is common. Hence multiple threads share a common global data region. Therefore, CreateThread() is not sufficient to replace fork() system call.

Along with CreateThread() , separate global data copy is to be provided to thread, so they can access there own copy. For this purpose one copy module is incorporated in the code separately named as ***copyglobal***. It is called just before CreateThread() call. It uses recursive function calls to copy sub data structures used in global data structure used by WinPVM daemon.

Solution to this problem we followed is as follows (in steps) (Please see fig. 3.1 and fig 3.2).:

- Convert complete global data in a huge structure.
- Make a Copy of Complete Global data (Before calling CreateThread())

- Call CreateThread()
- Attach child threadid to this copy of data
- Now child thread will modify its copy of global data and parent thread will access/update its copy. Any global data is accessed through its threadid.
- Just Before child Thread completion delete the copy of global data used by this thread. Life of this global data copy is till thread is active.

For this purpose some extra code is incorporated to PVM code. These files are CopyGlob.c, extralib.c . All functions required for above operations are kept in one of these two files only.

In some other places though fork() call is used but there is no global variable clash. In such cases fork() is replaced by CreateThread() and child process code with become Thread function. Parent code will run as it is.

One more point is here, which is related to fork(). It is closing of file, pipe and socket descriptor. In fork() environment if either parent or child process closes any descriptor, it will not affect . So normally if child does not require any descriptor, it closes it. The same time parent is using the same descriptor (closing has no effect on other part). In case of Windows 95 thread environment, if one thread closes any global descriptor, will not be available to other thread. Though we can replace fork() with CreateThread(), but if some such descriptor closing is there, do not close such descriptor. Such trick is used in Win95-PVM as well as in rshd also.

Over and above all of these issues some special requirements from Winsock library are also there. For such requirement please refer Appendix 2.

3.6 Asynchronous Communication using CreateThread()

In WinPVM console program, select call is used as polymorphic call, which provides asynchronous communication to network socket and file. In Windows95 select is only for socket. Due to non availability of select for read call , file read becomes blocking call. To avoid such blocking, two separate thread are created, one for network and other for console reading. In this way CreateThread() is also used to provide asynchronous communication.

Chapter 4

Implementation of WinPVM

WinPVM provides a heterogeneous distributed network computing environment like PVM [1], which also supports heterogeneity in OS. It can be viewed as a generalized extenuation to PVM; a tool for better and more cheaper network orchestration. In standard PVM, UNIX machines act as computing nodes. In WinPVM, UNIX Servers as well as Windows95 Systems can become computing node. A parallel virtual machine can be formed using PC running Windows95 as master node and other PCs or UNIX workstations as slave nodes. Alternatively UNIX workstation as master Pvmd node while other UNIX workstations or PCs running Windows95 as slave nodes.

This chapter describes the main implementation aspects of WinPVM, which distinguishes it from PVM[1].

WinPVM has three distinct modules, viz.

- PVM daemon,
- PVM console and
- PVM library interface for program development.

Functionally all the WinPVM components behaves identically the corresponding components of UNIX PVM.

4.1.1 WinPVM Daemon

WinPVM supports heterogeneity in OS, as shown in Fig 4.1, besides supplying Homogeneous Windows95 platforms as illustrated by Fig 4.2. Fig 4.3 describes an overview WinPVM system architecture.

The network communication protocol between daemons is always UDP and that between tasks or between tasks and daemons is always TCP. Fig 2.3 described earlier and Fig 4.1 together illustrate the network communication mechanism.

Windows95 is basically designed as a client to Windows NT. Therefore expectedly the authentication mechanism in Windows95 is very poor. Its authentication mechanism can not deny to access of the machine to anyone. A separate authentication layer is required to prevent unauthorized access of the machine over network. Some of network software like rsh, rshd (Windows 95 GUI shareware versions) concept to allow or deny a user from accessing the system. A similar authentication system has to be incorporated with WinPVM. But it must be noted that this layer is not secure enough to prevent others from using the PC, machine is not physically secured. A comparison of authentication and security mechanism between UNIX and Windows95 is depicted in Fig 4.4.

One major difference between WinPVM and UNIX PVM is in add host feature. In PVM implementation, it uses fork() is used to create slave pvmd from the master pvmd. The parent executes code relevant to it and sets one variable called ***runstate***. Similarly, child executes its code as pvmd′ and sets the same variable to a different value. In Thread environment all the related threads use the same set of global data as depicted earlier in Fig. 3.1.

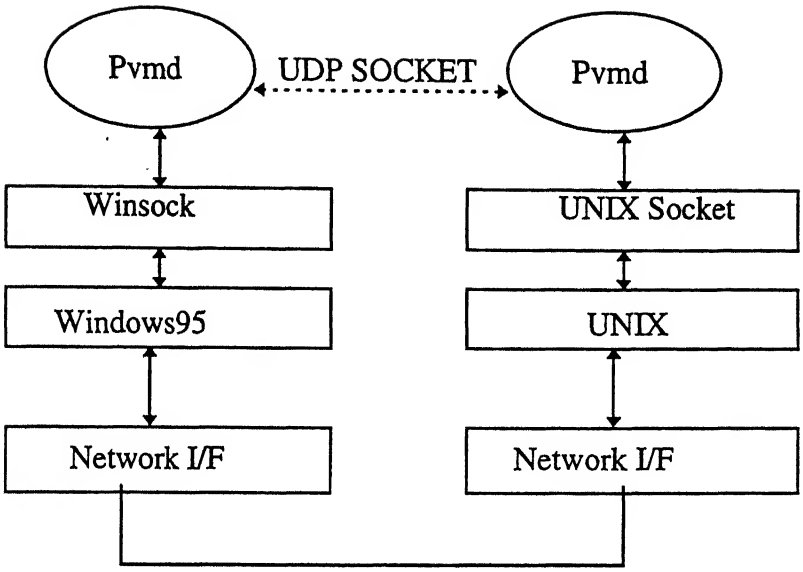


Fig 4.1: WinPvmd to UNIX Pvmd Connection
(Heterogeneous OS Support)

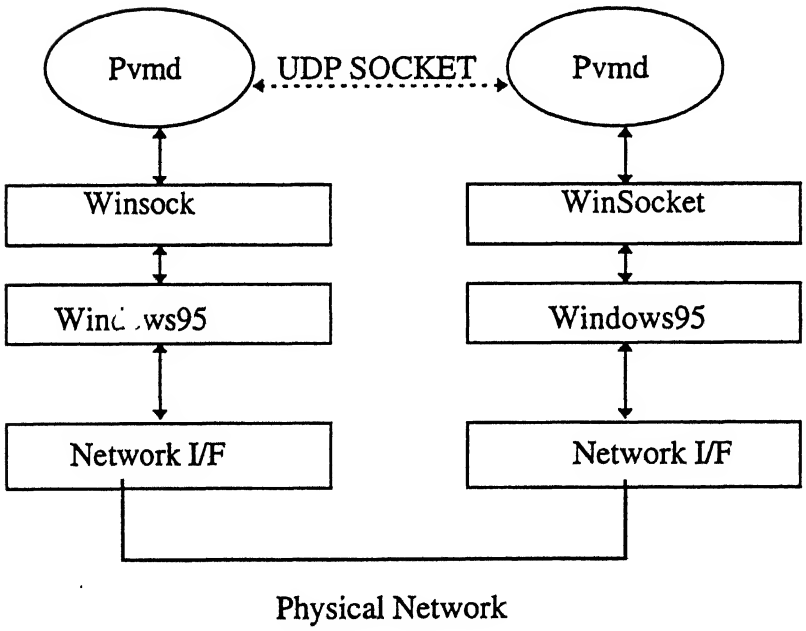


Fig 4.2: WinPvmd to WinPvmd Connection
(Homogeneous OS Support)

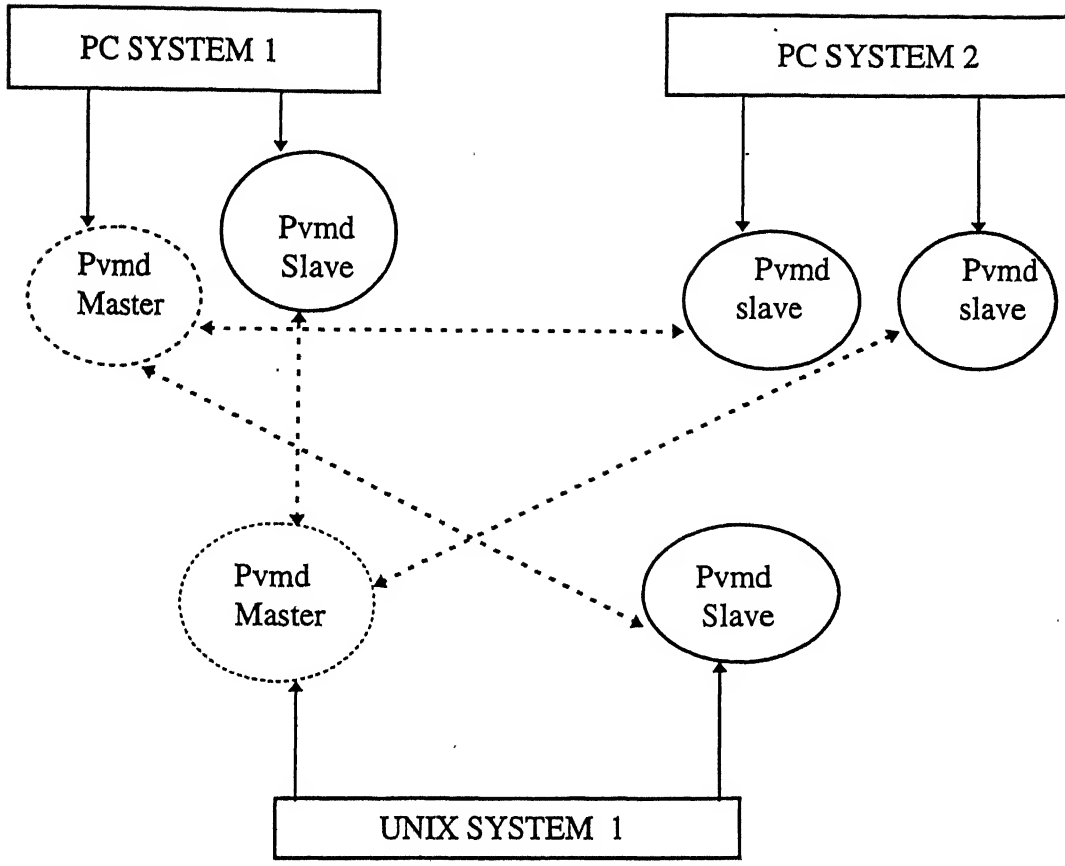


Fig4.3: Virtual Machines for WIN95-PVM and UNIX PVM can coexist

On the other hand both parent and child processes use separate copy of global data in case of fork() call (see Fig. 3.1 chapter 2). If parent modifies some variables, this modification will not affect child and vice versa.

Fork() call had to be simulate on CreateThread() to resolve the problem creating slave daemons in WinPVM from the master pvmd by preserving the functionality of the PVM. The detailed processing required for the fork() like function using threads below.

- Make a global structure for all global data.
- Initialize a global table which keeps thread id and pointer to global data for the thread.
- Before calling CreateThread, Create a replica of global structure.
- Fill global data table as < > <pointer to copy of global data>
- Call CreateThread() with pointer to global data copy
- Suspend child thread
- Fill global data table as <Thread id> <pointer to Copy of global data for the thread>.
- Enable child thread for execution.

Implementation of above algorithm provides required functionality of fork except for closing file/pipe/socket descriptor. In fork(), if either parent or child closes any such descriptor, it will not affect the other as shown in Fig. 4.4. In thread environment file descriptors are shared. Therefore, if any thread closes a descriptor, it will not be available to the other thread as in Fig 4.5. An easy way to overcome such problem is to close a descriptor only when it is ensured that none of the threads needs it.

Copy global data structure as required for fork() simulation is not a trivial problem. The complexity of the problem is mainly due to pointer variables. Recursive

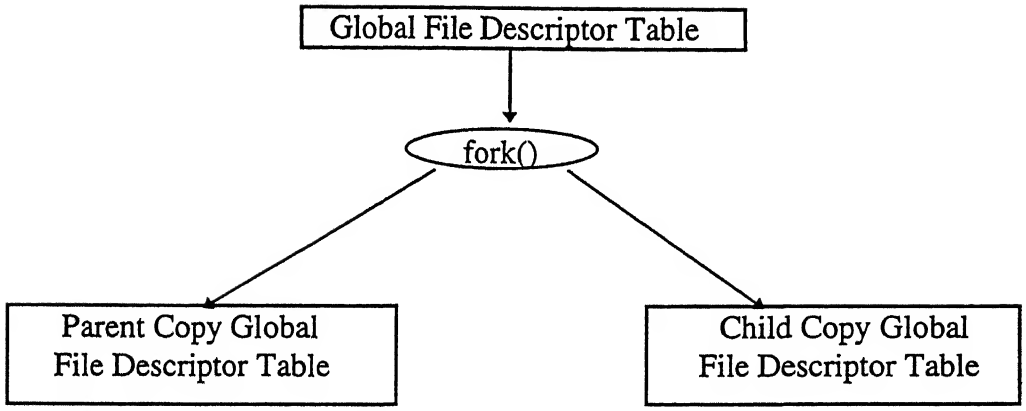


Fig. 4.4: fork()

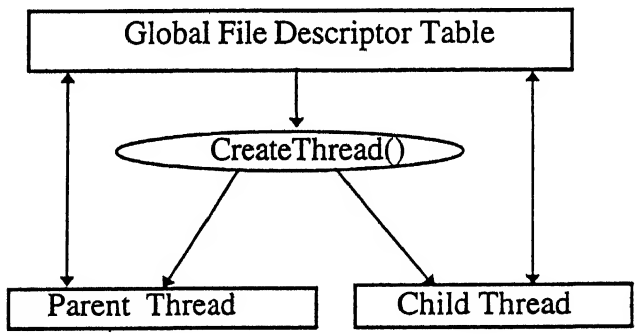


Fig. 4.5: CreateThread()

calls may be required to copy a sub data structure. It may be noted that global data structure consists of several variables and recursive sub structures due presence of pointers. Fig 4.6 illustrates the logic of copy routine which enables to keep consistency

with fork() semantics. The copy routine routine form almost one third of the WinPVM code.

UNIX PVM makes implicit use of polymorphism at many places. Several problems cropped up while resolving these polymorphism. Mainly read, write, close and select system calls employ polymorphism. As discussed in Chapter 2, one to one function replacement were possible in most cases. However, such replacements may have to adhere to call formats presented by Windows95. The formats may vary from call to call. Moreover, some UNIX system calls are not even available in Windows95. For instance select is not available for pipe and file. So, for WinPVM implementation we have been forced to use blocking ReadFile() instead of non blocking read call for pipe.

At present, only one PVM daemon can be run on a PC, either master pvmd or slave pvmd. However multiple slave/master daemons can run (as in UNIX node) if proper authentication (allocation of uid) layer similar to UNIX is incorporated with WinPVM. Fig 4.7 shows the differences in authentication system between UNIX and Windows95. Currently uid is assigned by setting a uid variable in autoexec.bat

Another problem is due to use of file descriptor as socket descriptor or socket descriptor as file descriptor. During add host, pvmd' (pvmd prime) sends an EOF to slave pvmd at the end. In slave pvmd this EOF is read from stdin. Such intermixing in operations is not available in Windows95. So WinPVM ignores EOF as illustrated by Fig 4.8.

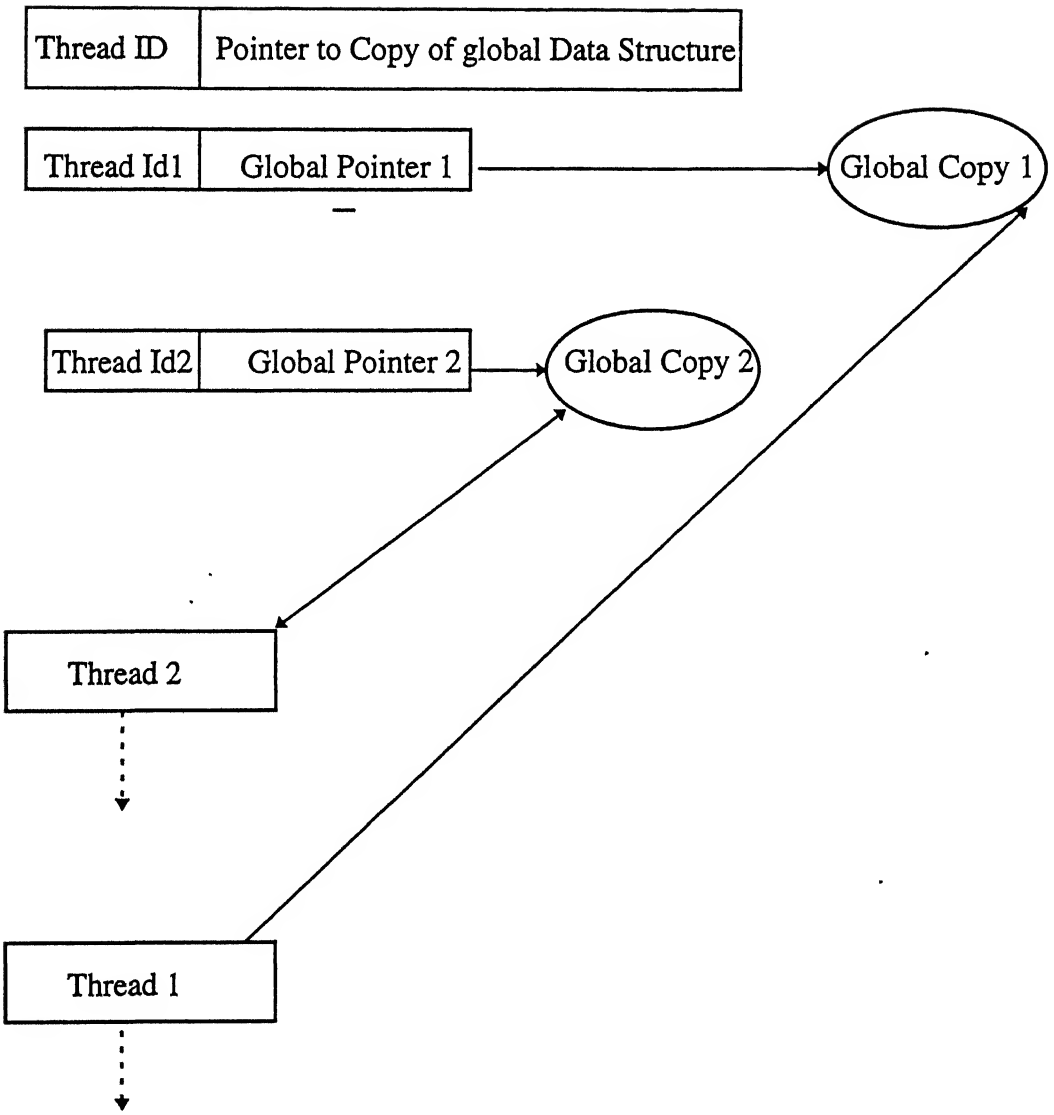
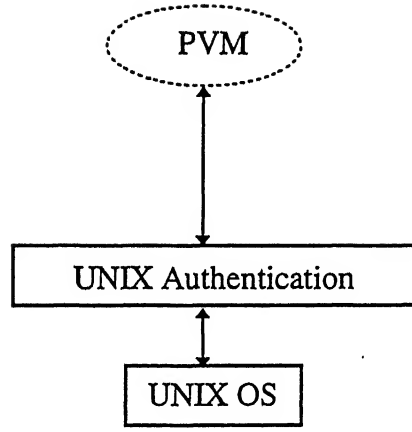
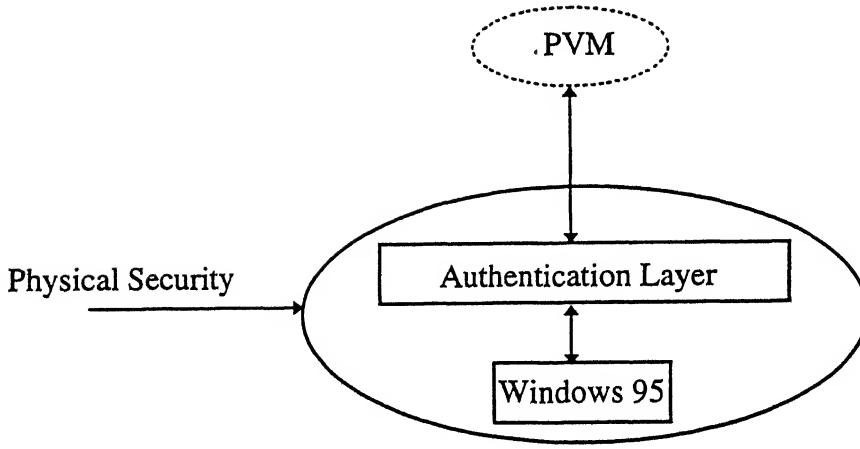


Fig 4.6: Global Structure Thread Table

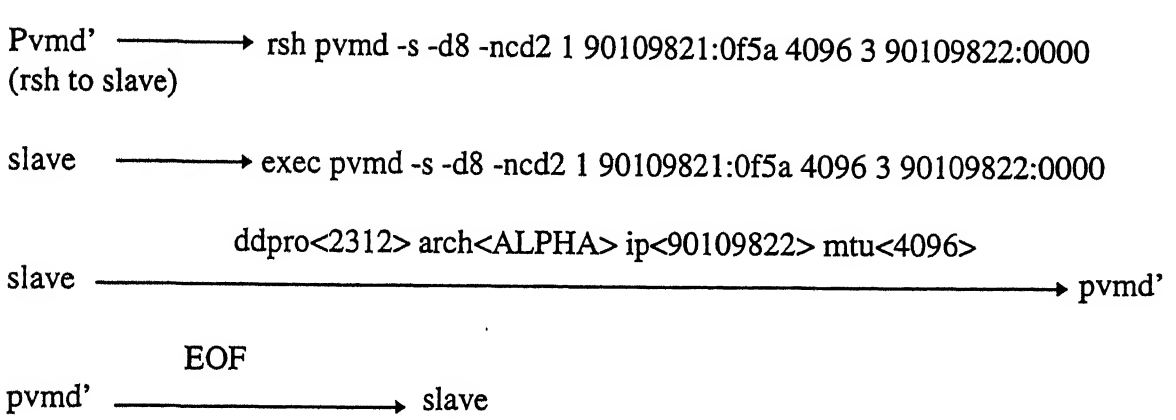


UNIX Authentication Layer

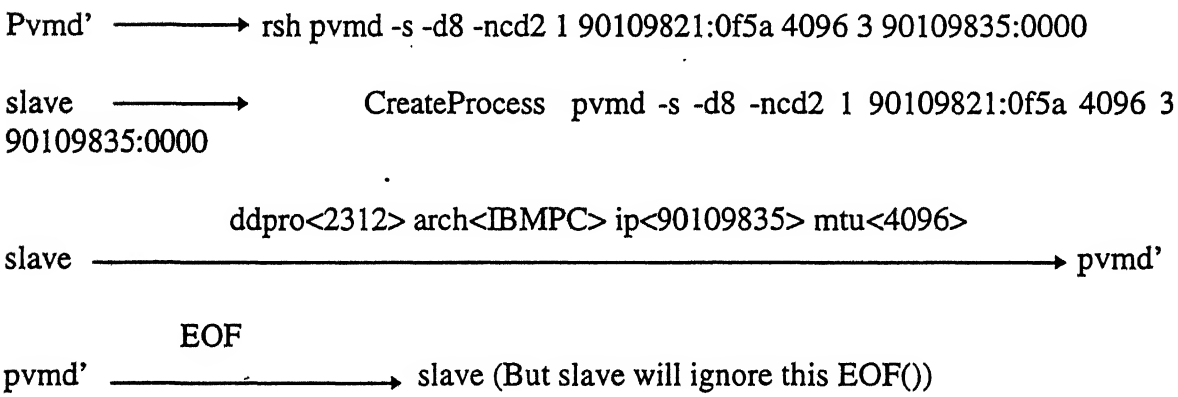


Windows 95 Needs Physical Security to implement Authentication Layer

Fig 4.7



UNIX to UNIX addhost protocol



UNIX to Windows95 and Windows95 to Windows95 addhost protocol

Fig 4.8 Difference in working of routine addhost()

4.1.2 WinPVM console

PVM console is a program which is used as a manager to virtual machine. As discussed earlier it provides all such services required to mange virtual machine. WinPVM console and UNIX PVM console have same functionality. But internal design is quite different.

In UNIX PVM console select system call is used as polymorphic system call as shown below.

```
While (1)
        {
                select() call for socket and stdin
                if (FD_ISSET(stdin))
                        read(0,....)
                }
                do(cmd);

        }
```

It is used for notification for reading of a file (console is a file in UNIX) as well as for a network socket. It helps to use a blocking call as an asynchronous call. Such polymorphism is not provided by select function in Windows95. *Select()* works only for network sockets. So select call not be used as it is in Windows95. Due to this limitation, read file can not be used as it is. In order to simulate the similar asynchronous environment, *CreateThread()* as shown in Fig 4.9 is used twice. The first *CreateThread()* creates a thread, which execute network read. The second *CreateThread()* creates another thread which is used to read console (stdin) only.

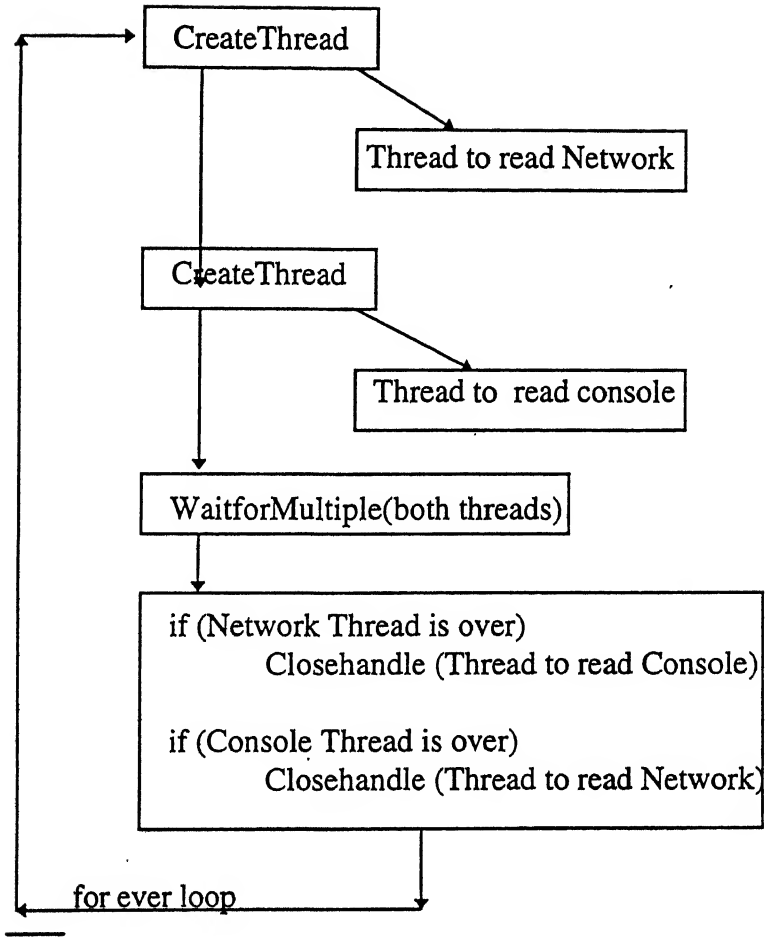
After creating the threads, the parent thread waits for completion of either of the child threads. If thread for network read is completed first, it kills other thread, which reads console and vice versa. However, read file call is a blocking call, but it does not affect other code to execute asynchronously.

Initially WSAStartup() call is used In WinPVM console (WSAStartup is a mandatory call for programs using Winsock library). As explained earlier, in WinPVM console two separate threads are used in addition to main thread to provide asynchronocity in file *read()*.

4.1.3WinPVM library support

In UNIX PVM library interface is available for C and FORTRAN. However WinPVM library interface is available for C only at present. This library is fully compatible with UNIX PVM library for C supporting exactly identical primitives. The only issue is, along with this library, user has to include XDR and Winsock32 libraries also. The changes required to run PVM application provided in appendix 2.

Some of the functions used in UNIX PVM are not available in Windows95 environment. For such function like *gettimeofday(), ffs()* etc., a separate library, called extralib, is created. This library is used by PVM daemon and PVMlib.




```
Function Network  Thread ()
        {
                Select ( network socket descriptor);
        }

Function Console Read()
        {
                read(stdin, ...)
                docnd(cmd);
        }
```

Fig 4.13: WinPVM Console main loop along with two thread functions

Chapter 5

Conclusion

The development of Win95-PVM is a step towards low cost parallel computing. Win95PVM opens opportunities to use Windows95 based PCs for parallel computing. Of course PCs running Linux or BSD386 could used. However, the OS requirement for running PVM remains either UNIX or UNIX clones.

During this development, a framework emerged for porting any UNIX based network software to Windows 95 platform. Using this framework one can port most of the UNIX based software in to Windows 95 platform. For example UNIX base implementation of MPI can be easily ported to Windows95 under this framework.

Using porting tips a user assisted automated tool for porting programs from UNIX to Windows95 can be developed. Such a tool may be extremely useful when an establishment decides to switch for a better OS environment Infect, the framework may provide guidelines to develop an object oriented approach with explicit use of OS independent software. Polymorphism in function and system calls will easily support OS heterogeneity.

XDR library, rsh and rshd daemon on Windows95 are the byproducts of porting issues. Now other users can use these utilities.

Though Win95PVM is operational now it has certain drawbacks. The authentication layer is missing. Another point is regarding application development. To incorporate Winsock and Windows program some standard code is patched in every application. The patching can be done in a better way. WSAStartup() and WSACleanup() call can be put inside pvm library. For

example, for WSAStartup() a standard routine pvm_wsockstartup() can be made. Similarly for WSACleanup() can also be handled..

WinPVM is running successfully, but internally one has to clean the code as well as proper documentation is required. WinPVM is working smoothly in experimental setup. But some users may face problem during use. Such problems ,if any, can be rectified later.

Appendix 1

Setup and configuration of Windows 95 PVM

The Windows 95 PVM is organized as follows. C:\pvm is the root directory for the entire system. The root directory has number of other directories viz.,

src	Source Code for pvmd.exe and libpvm.lib
console	Source Code for console program pvm.exe
examples	Source code for application and example programs
include	Header files required for compilation of PVM programs
xdr	Source code for XDR Library
doc	Document for Windows95 PVM
rsh	Source Code for rsh.exe (NCSA)
rshd	Source Code for rshd.exe(BDS 4.3)
lib	Contains only Libpvm.lib and XDR.LIB
bin	All application executable along with executable for PVM console, pvmd daemon, rsh and rshd are located here.
etc.	stores hosttype file

Certain environment variables have to be set for Win95-PVM. These variables can be included in c:\autoexec.bat. This ensure the setup is ready when the PC boots up. The specific settings are as follows.

```
SET TMP=c:\tmp
SET HOME=c:\
SET PVM_ROOT=C:\pvm
SET PVM_ARCH=IBMPC
SET UID=<uid>   {In UNIX SYSTEM}
```

Apart from these system specific environment one PVM host configuration file should be made available. This file indicates pvmd path, working directory path and directories for executables for the other possible physical nodes which may constitute the parallel virtual machine. A sample pvmhost file is given below.

```
#PVM hostfile with options.
#wd specifies the working directory for each machine.
# If ep is not specified,
#$(HOME)/pvm3/bin/$(PVM_ARCH) is searched for the executable.


#set default options for the following hosts.
*dx=/usr/local/pvm/lib/ALPHA/pvmd3
*ep=$HOME/pvm3/bin
&csealpha1
&csealpha2
#set pvmd path and executable path for csealpha3
&csealpha3 dx=/3d6/pdcs/rkg/pvm/pvm3/lib/ALPHAMP/pvmd3
 ep=/3d6/pdcs/rkg/pvm/pvm3/bin/ALPHAMP
#set pvmd path and executable path for pc39, pc53
&pc53  dx=d:\pvm\bin\pvmd.exe  wd=c:\pvm\bin  ep=\\pvm\\bin
&pc39  dx=c:\pvm\bin\pvmd.exe wd=\pvm\bin ep=\\pvm\\bin\\
```

Appendix 2

Application Programs in Win95PVM

In order to run PVM programs under windows95 platform. some changes have to incorporate in these programs. In case of Windows95 Winsock based network programming services. WSAStartup() function must be called to start socket programming. Similarly WSACleanup() is also mandatory for network cleanup before termination of a program. In addition <windows.h> must be included in any Windows program. For the sake of completeness, the details of required changes to run PVM hello.c are provided below.

A.1 UNIX based PVM Hello.c

```
#include <stdio.h>
#include "pvm3.h"

main()
{
        int cc, tid;
        char buf[100];

        printf("i'm t%x\n", pvm_mytid());

        cc = pvm_spawn("hello_other", (char**)0, 0, "", 1, &tid);

        if (cc == 1) {
                cc = pvm_recv(-1, -1);
                pvm_bufinfo(cc, (int*)0, (int*)0, &tid);
                pvm_upkstr(buf);
                printf("from t%x: %s\n", tid, buf);

        } else
                printf("can't start hello_other\n");

        pvm_exit();
        exit(0);
}
```

A.2 Windows 95 based Hello.c program

```
// Inserted portions of codes for Win95-PVM are in different font
#include <windows.h>
#include <winsock.h>
#include <stdio.h>
#include "pvm3.h"

#define MAJOR_VERSION       1
#define MINOR_VERSION       2
#define WSA_MAKEWORD(x,y)           ((y) * 256 + (x)) /* Hl:Minor, LO:Major
*/

char * _calloc (unsigned , unsigned);

main()
{
        int cc, tid;
        char buf[100];
        int ret;
        WORD VersionReqd;
        LPWSADATA lpmyWSAData;
        WORD wMajorVersion, wMinorVersion;


 wMajorVersion = MAJOR_VERSION;
 wMinorVersion = MINOR_VERSION;
 VersionReqd=WSA_MAKEWORD(wMajorVersion, wMinorVersion);
 lpmyWSAData = (LPWSADATA)_calloc(1, sizeof(WSADATA));
 ret = WSAStartup(VersionReqd, lpmyWSAData);

        printf("i'm t%x\n", pvm_mytid());

        cc = pvm_spawn("hello_other", (char**)0, 0, "", 1, &tid);

        if (cc == 1) {
                cc = pvm_recv(-1, -1);
                pvm_bufinfo(cc, (int*)0, (int*)0, &tid);
                pvm_upkstr(buf);
                printf("from t%x: %s\n", tid, buf);

        } else
                printf("can't start hello_other\n");

        pvm_exit();
        exit(0);
}


char *
```

```
calloc (nelem, elsize)
unsigned nelem, elsize;
{
        HANDLE hMem;
        PSTR ptr;
        unsigned size = nelem * elsize;

        if ((hMem = LocalAlloc(LPTR, size)) == NULL)
                return (char *) 0;
        if ((ptr = LocalLock(hMem)) == NULL) {
                LocalFree(hMem);
                return (char *) 0;
        }
        return (char *) ptr;
}

void
free (void *cP)
{
        (void) LocalFree(LocalHandle((WORD) cP));
}
```

Similar changes are required to convert any UNIX PVM application into Windows95 PVM Application.

References

[1] Al Geist, Adam Beguillin, Jack Dongarra, Weicheng Jiang, Robert Mancheck, Vaidy Sunderam, PVM: Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing, 1994,The MIT Press, Cambridge, Massachusetts, London, England.

[2] R. Butler and E. Lusk. Monitors, messages and clusters: The P4 parallel programming system. Technical Report Preprint MCS-P362-0493, 1993, Argonne National Liboratory, Argonne, IL.

[3] J. Flower, A. Kolawa and S. Bharadwaj. The Express way to distributed processing. Supercomputing Review, page 54-55, May 1991.

[4] Message Passing Interface Forum. MPI: A message-passing interface standard. Computer Science Dept. Technical Report CS94-230, University of Tennessee, Knoxville, TN, April 1994.

[5] Nicholas Carriero and David Galernter. LINDA in context. Communications of the ACM, April 1994

[6] James L. Conger, Windows API Bible, 1994,Galgotia Publications Pvt. ltd New Delhi.

[7] Brain Myers - Eric Hammer, Windows NT Programming , 1993,Sybex/Tech Asian Edition.,Tech Publications Pte Ltd, Singapore

[8] Marc J. Rochkind, Advanced UNIX Programming, 1985,Printice-Hall Software Series, Printice-Hall Inc, Englewood cliff, New Jersey 07632 (Printed in USA)

[9] Ralph Davis, Windows Network Programming, 1992,Addison-Wesley Publishing Company, Reading Mass.

[10] Paul Dilascia, Windows++ ,1993,Addison-Wesley Publishing Company, Reading Mass.

[11] W. Richard Stevens, UNIX Network Programming,1992, Printice-Hall of India Pvt. Ltd., New Delhi

[12] Martin Hall-Mark Towfiq- Geoff Arnold- David Treadwell- Henry Sanders, Winsock API 1.1 Reference .

[13] Alexandre Alves, Luis Silva , Jao Carreira, Joao Gabreif Silvia ,WPVM Parallel Computing for People, Departmento de Engenharia Informatica, Universidade de Coimbra - Portugal

S/W:

[13] PVM 3.3 Source Code, www.epm.ornl.gov/pvm/pvm_home.html

[14] RSHD Source Code [BSD 4.3], ftp.cs.berkely.edu

[15] rsh Source Code [BSD 4.3] and [NCSA], www.ncsa.uiuc.edu

[16] XDR Library Source Code [SUN], ftp.sunlabs.com